CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CE144:OBJECT ORIENTED PROGRAMMING WITH C++

UNIT 10

Managing Console I/O Operations

DEPSTAR Devang Patel Institute of Advance Technology and Research

Content

- Introduction
- C++ Stream
- C++ Stream Classes
- Formatted and Unformatted Console I/O Operations

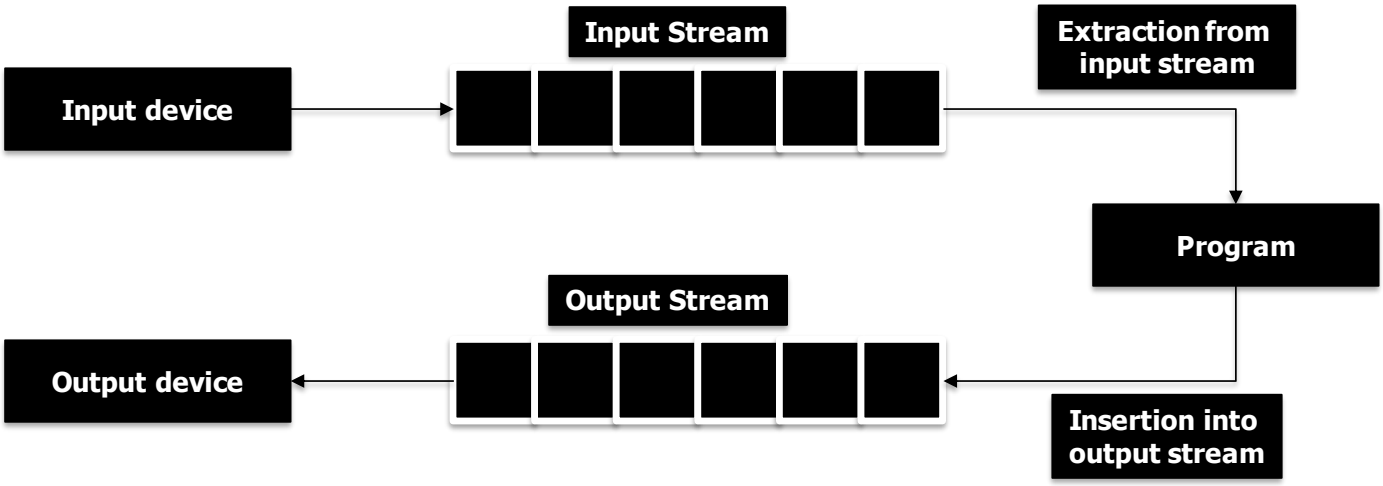
Introduction

- C++ uses the concept of stream and stream classes to implement its I/O operations with the console and disk files.
- C++ support all of C's rich set of I/O functions.

C++ Stream

- Stream is an interface supplied by the I/O system of C++ between the programmer and the actual device being accessed.
- It will work with devices like terminals, disks and tape drives.
- A stream is a sequence of bytes.
- It acts either as a source from which the input data can be obtained or as a destination to which the output data can be sent.

C++ Stream


Input Stream
Extraction from input stream
Input device
Program
Output Stream
Output device
Insertion into output stream

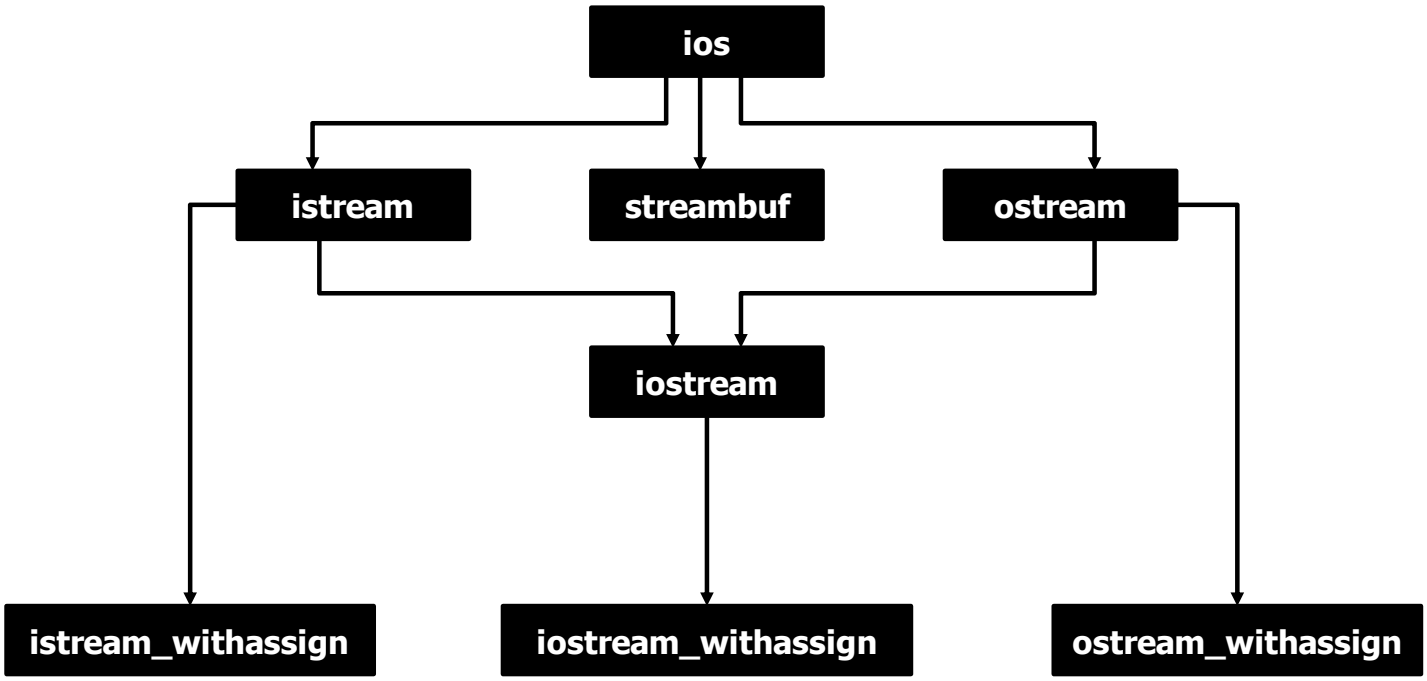
C++ Stream

- Input Stream - The source stream that provides data to the program.
- Output Stream - The destination stream that receives output from the program.
- The data in the input stream can come from keyboard or any other storage device.
- The data in the output stream can go to the screen or any other storage device.

C++ Stream Classes

- The C++ I/O system contains a hierarchy of classes that are used to define various streams to deal with both the console and disk files.
- These classes are called stream classes.
- These classes are declared in the header file **iostream**.

C++ Stream Classes



C++ Stream Classes

- Ios: Provides the basic support for formatted and unformatted I/O operations.
- Istream : Provides the facilities for formatted and unformatted input
- Ostream : Provides the facilities for formatted output
- Iostream : Provides the facilities for handling both input and output streams.

Unformatted I/o Operations

put( ) and get( ) Functions

- member functions of istream and ostream classes.
- For single character input/output operations.
- There are two types of get( ) functions:
  - get(char*) → Assigns the input character to its argument.
  - get(void) → Returns the input character.
    - char c; cin.get(c) c = cin.get( );
  - put( ) → used to output a line of text, character by character.
    - char c; cout.put('x'); cout.put(c);

getline( ) and write( ) Functions

- getline( ) function reads a whole line of text that ends with a newline character.
- cin.getline(line, size);
- Reading is terminated as soon as either the newline character '\n' is encountered or size-1 characters are read.
- write( ) function displays an entire line of text.
- cout.write(line, size);
- write( ) also used to concatenate strings.

Formatted I/o Operations

C++ supports a number of features that could be used for formatting the output.

These features include:

- ios class functions and flags.
- Manipulators.
- User-defined output functions.

Ios class functions

width( ) → to specify the required field size for displaying an output value.

precision( ) → to specify the number of digits to be displayed after the decimal point of a float value.

fill( ) → to specify a character that is used to fill the unused portion of a field.

setf( ) → to specify format flags that can control the form of output display.

unsetf( ) → to clear the flags specified.

Manipulators

Manipulators are special functions that can be included in the I/O statement to alter the format parameters of a stream.

To access manipulators, the file iomanip.h should be included in the program.

- setw( )
- setprecision( )
- setfill( )
- setiosflags( )
- resetiosflags( )

Difference between Manipulators & ios Member Functions

- The ios member function return the previous format state which can be used later.
- But the manipulator does not return the previous format state.

```
cout.precision(2);   // previous state.
int p =cout.precision(4); // current state, p=2.
cout.precision(p); // change to previous state
```

Designing Our Own Manipulators

- We can design our own manipulators for certain special purposes.

```
ostream& manipulator_name (ostream& output)
{
    …….  (code)
    return output;
}
ostream& unit (ostream& output)
{
    output << "inches";
    return output;
}
cout << 36 << unit;          → will produce "36 inches".
```

Designing Our Own Manipulators

We can also create manipulators that could represent a sequence of operations:

```
ostream & show (ostream & output)
{
   output.setf(ios::showpoint);
   output.setf(ios::showpos);
   output << setw(10);
   return output;
}
```

This function defines a manipulator called show that turns on the flags showpoint and showpos declared in the class ios and sets the field width to 10.


Thank you!